SLAM
Empirical Analysis

EDAA - G06

Henrique Ribeiro – Tiago Duarte
João Costa
João Martins

Octomaps/Octrees Analysis

Octomaps/Octrees In Our Project

- The octrees used in the max depth 16;
  - This depth leads to a 8^16-1 = 2.8147498e+14 nodes.
- Resolution of 1 cm;
  - This resolution with the amount of nodes available lets up map a volume of 655 m^3.
- The **octomap** will be **probabilistic**:
  - 3 types of cells: free, occupied, and unknown;
  - Each cell has a probability of being empty (**log-odds**)
    - > 0 more likely to be occupied
    - < 0 more likely to be empty
  - Unknown cells are uninitialized

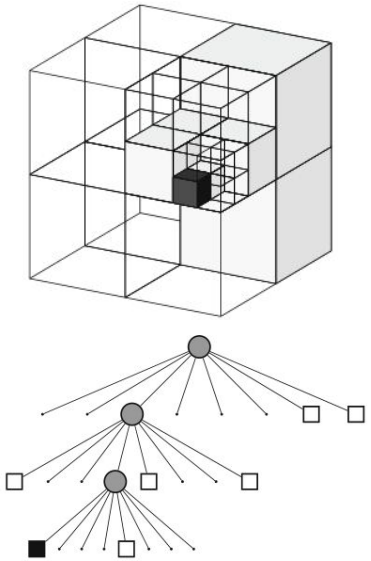


Fig 1. Example of octree storing occupied (black) and free (white) cells (Hornung et al.)

Octree Pruning

- If all child nodes are stable and agree on something, these nodes can be pruned, and their value goes to the parent node.

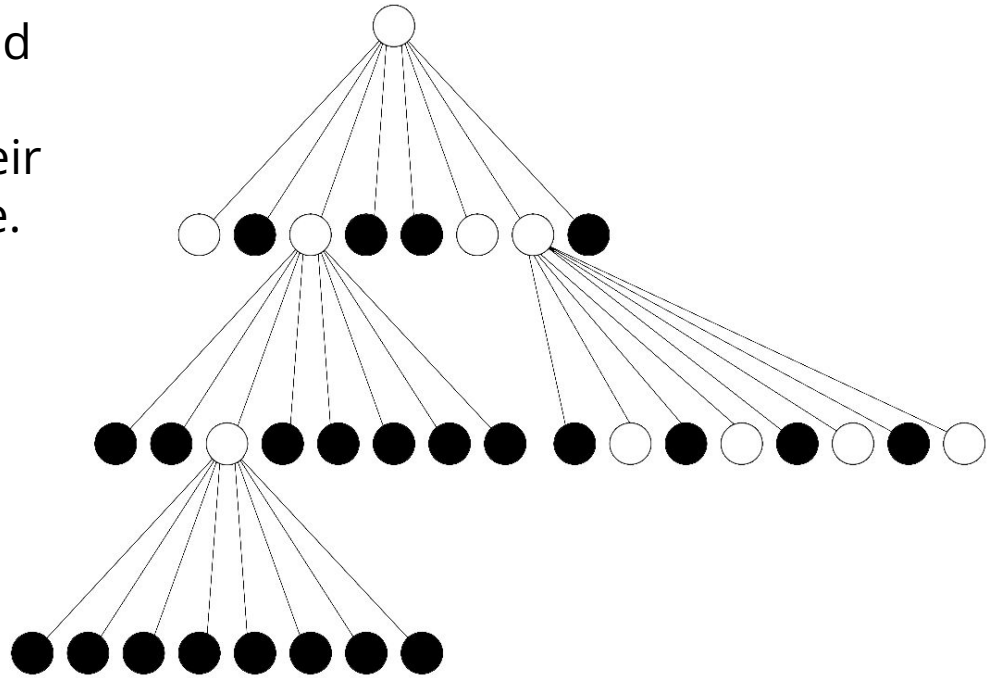


Fig 2. Simple octree representation

Octree Pruning

- This check is recursive starting from the leaves, and additional prunes can happen.

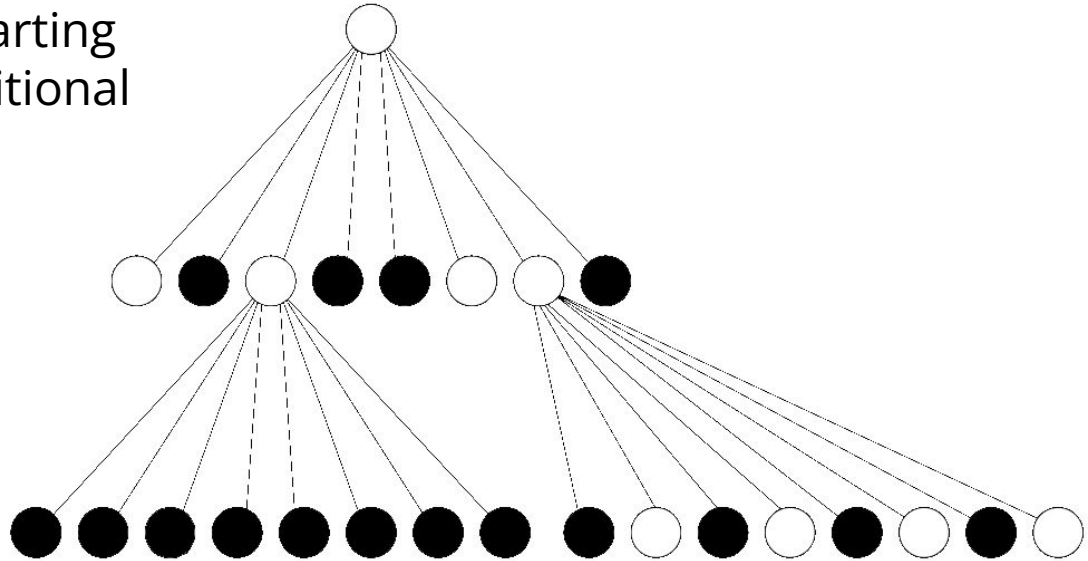


Fig 3. Simple octree after pruning

Octree Pruning

- Since no more nodes have all children in agreement, this layout would be the final one for this specific octree.

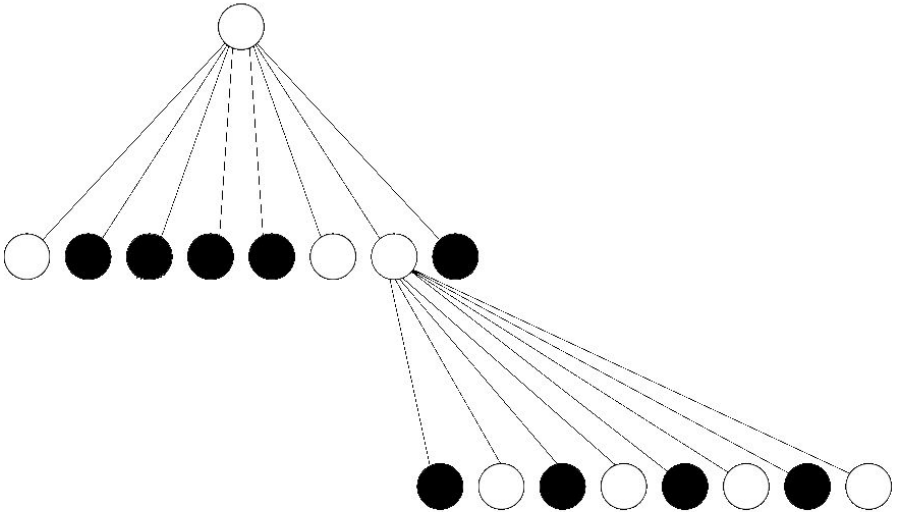


Fig 4. Simple octree after second pruning

Lazy Eval

- Used when a lot of nodes are updated at the same time.
- Does not update parent nodes when inserting children.
  - Saves updating until the end of all insertions.

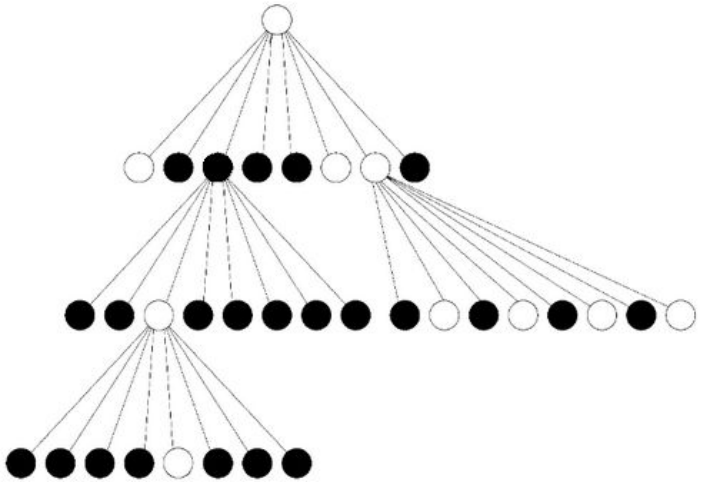


Fig 5. Unupdated octree

Octree Pruning In Action

- The execution time decreases by almost 2 times between lazy eval and no pruning, whereas with pruning it takes more time;
- The number of nodes decreases by almost 58-fold;

	With Pruning	Without Pruning	Lazy Eval.
Number of Nodes	19 769	1 145 977	19 769
Average Execution time of 5 runs	13.8s	5.31s	2.9s
Number of Intermediate Nodes	5 201	145 977	5 201
Number of Leaf Nodes	197 69	1 000 000	19 769

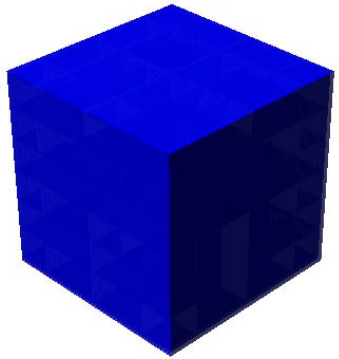


Fig 6. Cube 100cmx100cmx100cm

Keeping Children In Octomaps/Octrees

- Instead of keeping 8 pointers to each child node, an array of 8 children is kept.
- This mean that leaves only have one null pointer instead of 8 null pointers.

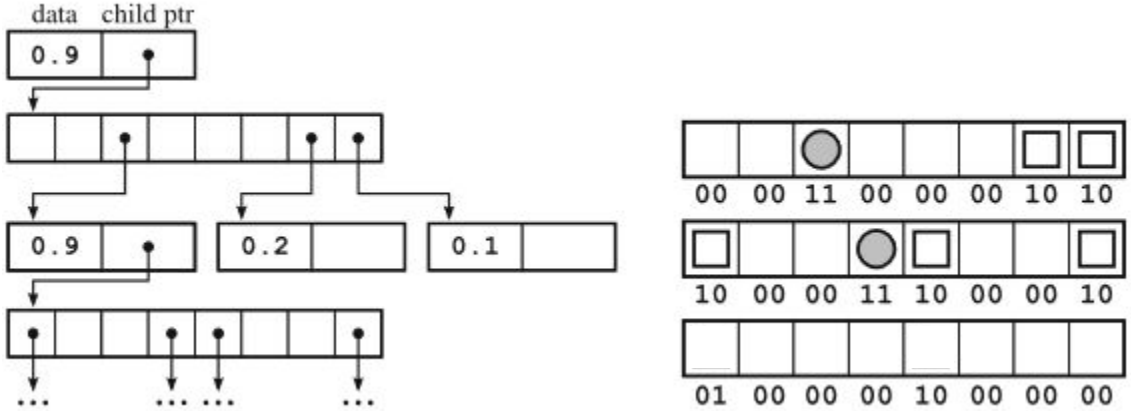


Fig 7. Comparing array to eight pointers (Hornung et al.)

Keeping Children In Octomaps/Octrees

- Using the previous measurements of 19 769 leaf nodes after pruning, when considering a 64-bit machine:

	Using Array	Using 8 Pointers
Number of Null Pointers	19 769	158 152
Structure size (in bytes)	158 152 Bytes	1 265 216 Bytes

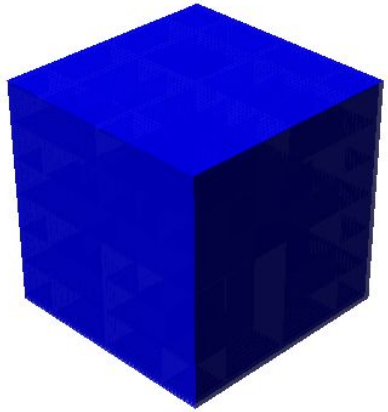
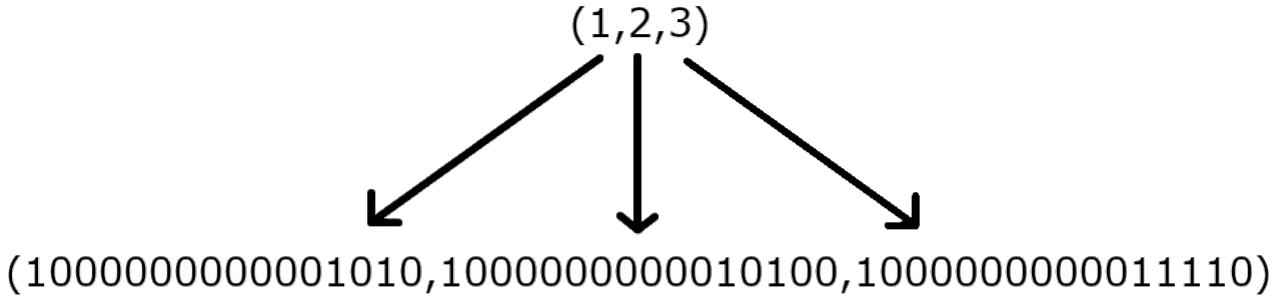


Fig 8. Cube 100cmx100cmx100cm

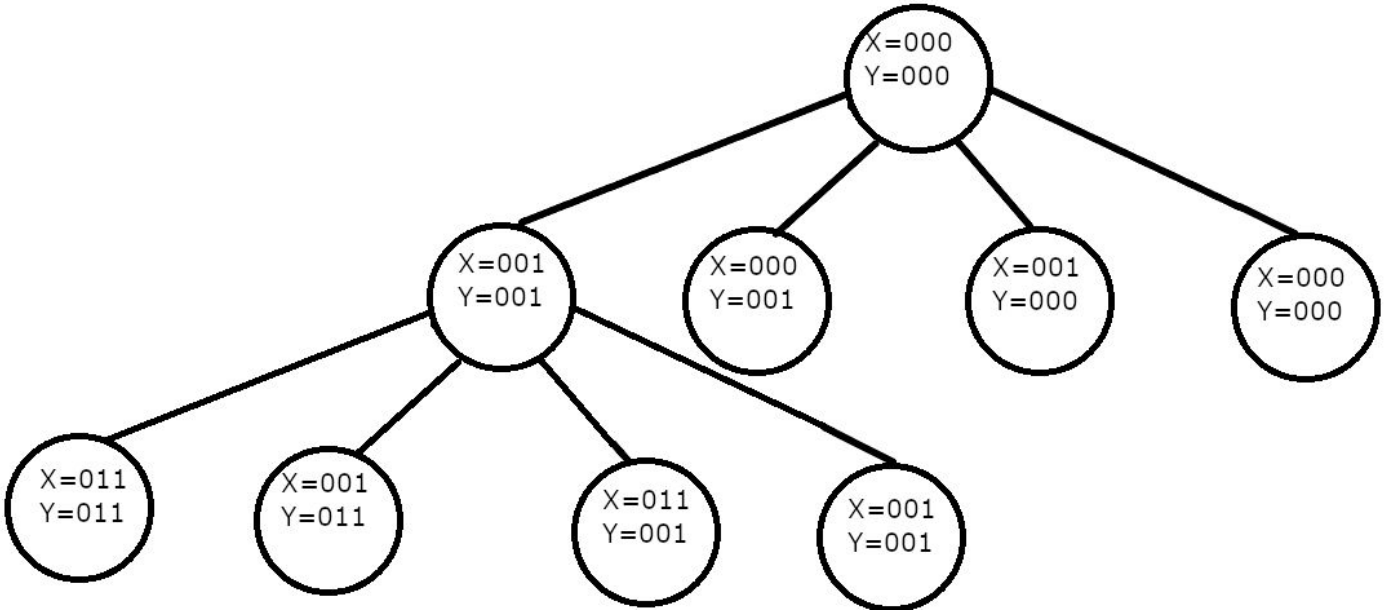
Octree Keys

- Instead of keeping the coordinates, each node computes a key value for x, y and z, each containing the same number of bits as the max depth.
- Every node has the first N bits of each key set, where N is the depth of the node, the other bits are set to 0.
- With these keys, for our usage of **octomaps**, the look-up is O(1).
- There is no direct correlation between coordinates and key values (For example, two sequential coordinates may not have sequential keys).



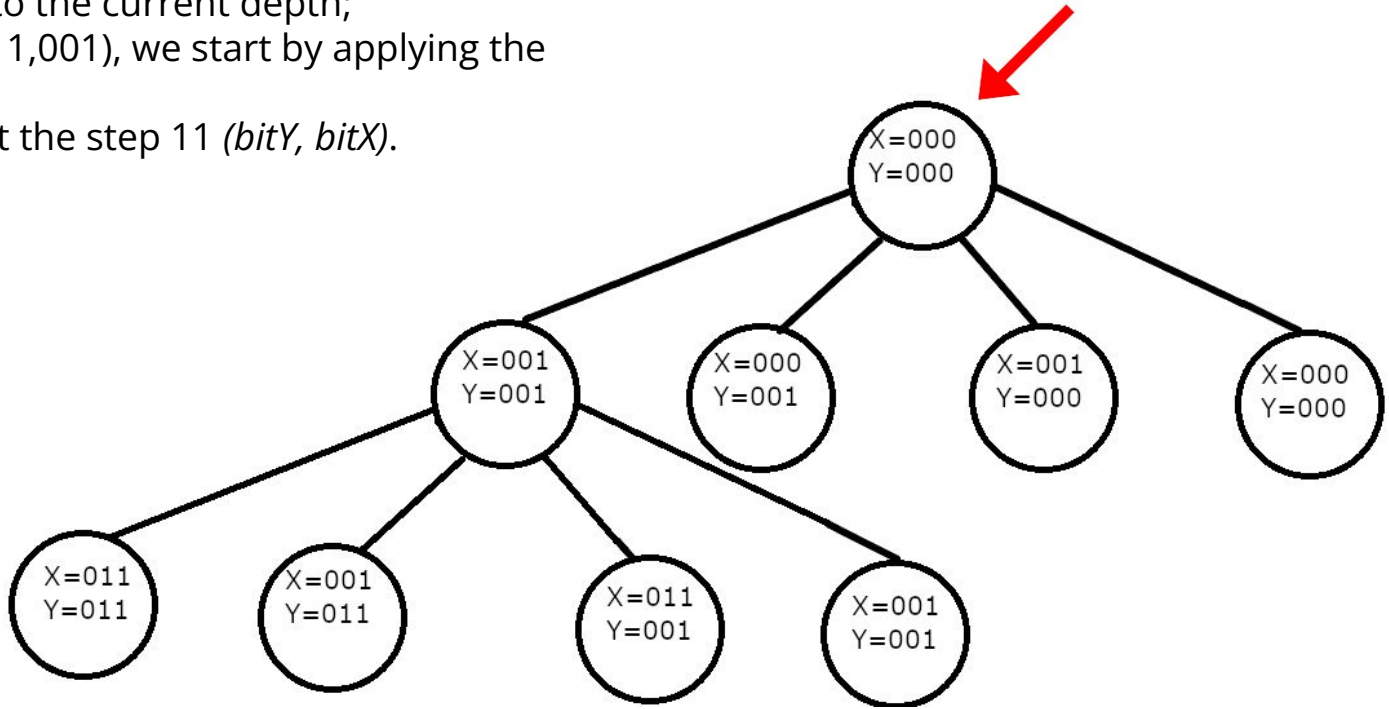
Look up using Octree Keys

- Assuming we are looking for coordinate (2,1), and it's key value is computed to (011,001).



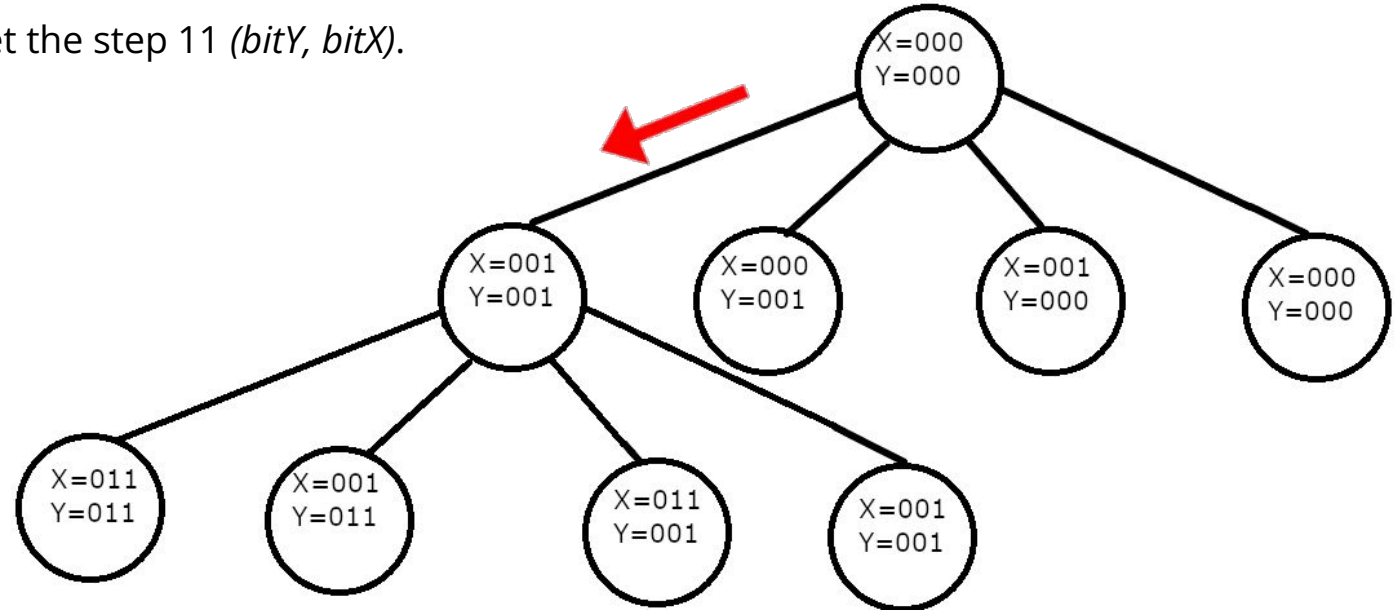
Look up using Octree Keys

- Starting at the root node, we see the bits at the position equal to the current depth;
- For the keys (011,001), we start by applying the mask 001.
- With this we get the step 11 *(bitY, bitX)*.



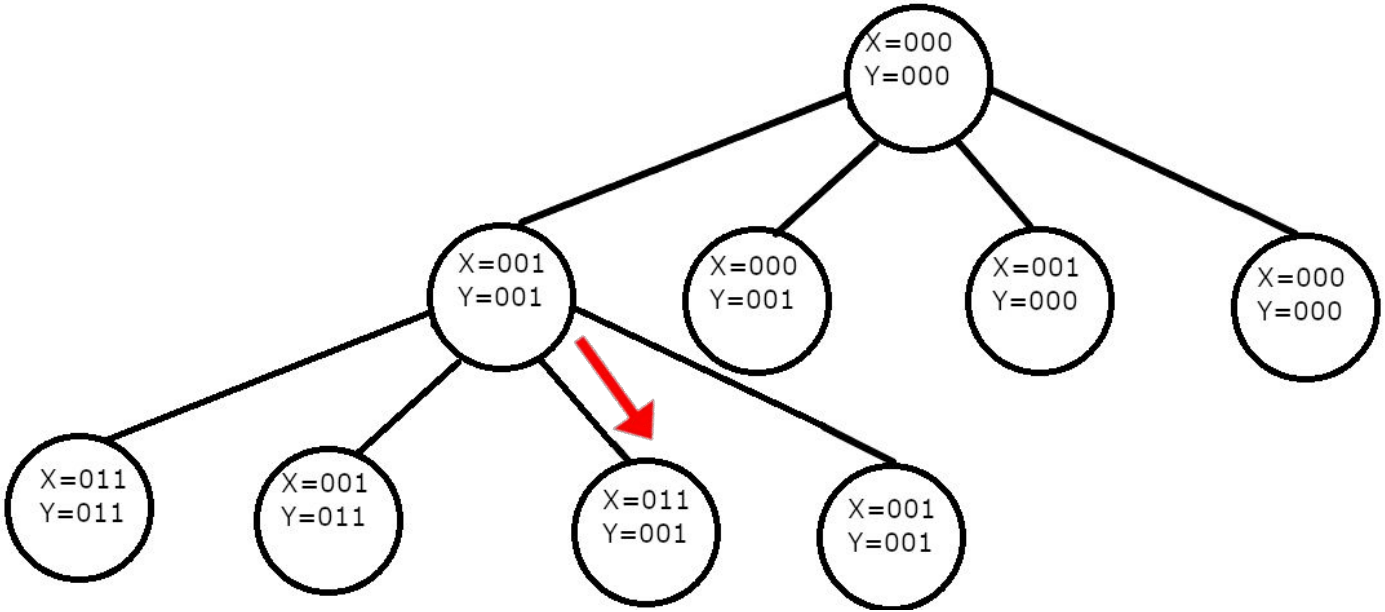
Look up using Octree Keys

- Starting at the root node, we see the bits at the position equal to the current depth;
- For the keys (011,001), we start by applying the mask 001.
- With this we get the step 11 *(bitY, bitX)*.



Look up using Octree Keys

- The next step, we use the mask 010.
- For the keys (011,001), after applying the mask we obtain the step 01.



Node Values Update Analysis

Updating Intermediate Log Odds

- Updating a leaf node can change all intermediate nodes before it, except if:
  - The leaf is already stable
  - Log odds update of 0 (irrelevant)
- Always updating the nodes is not desirable, as some values may not be changed.
- **Performing a search beforehand** to verify if the update is relevant can mitigate this issue.

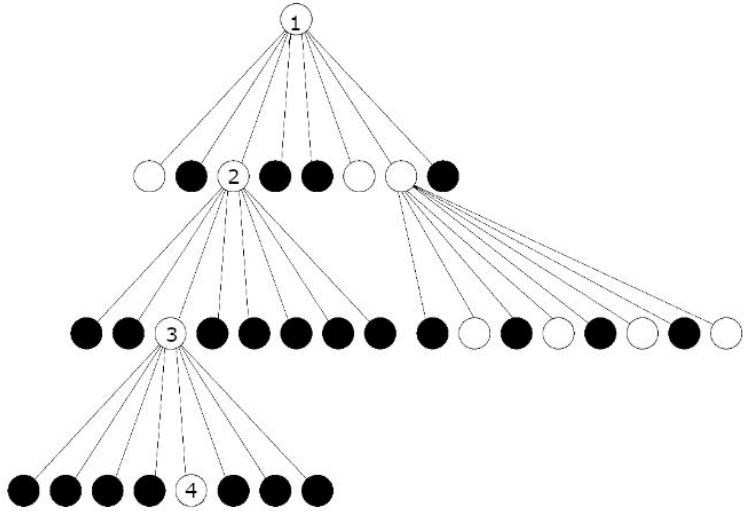


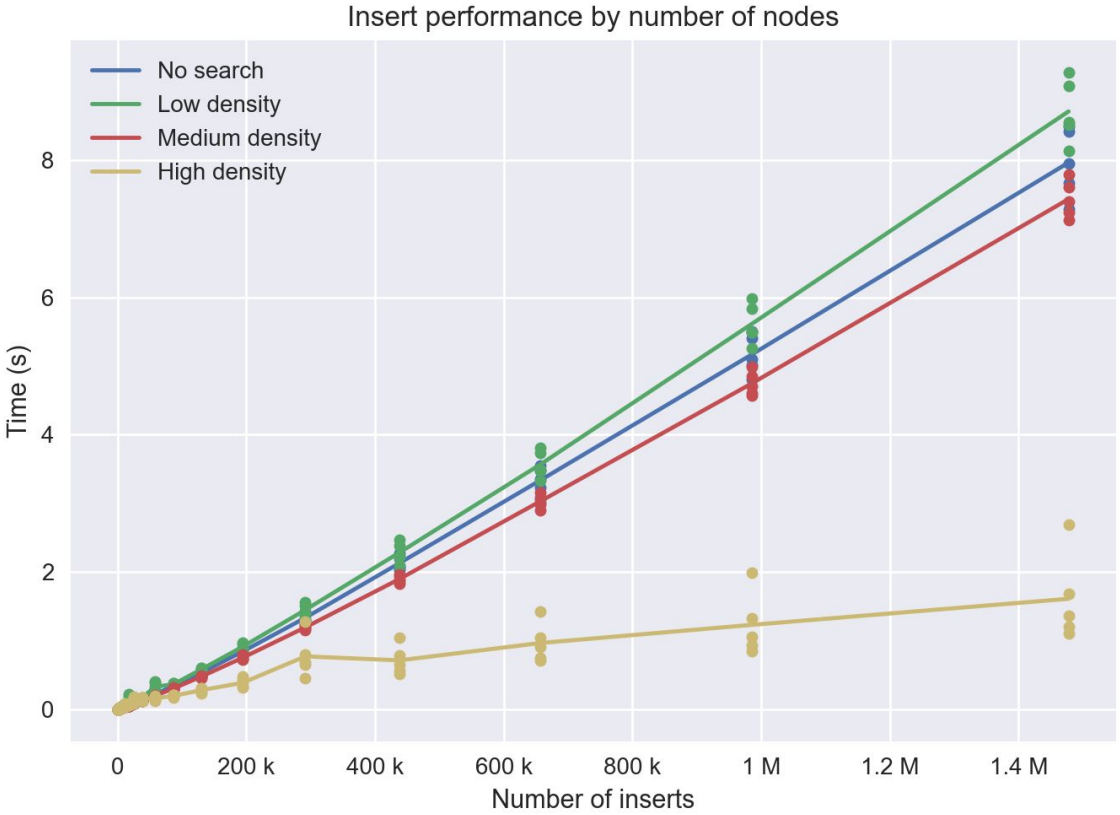
Fig 9. Octree with numerated nodes

Updating Log Odds Values

	With Check	Without Check
Number of Updates	10 000 000	10 000 000
Average Execution time	4.5s	10.5s

- 43% performance boost when using the check before updating the values.

Comparing Results



Points followed a normal distribution with varying std:

μ = 10000
low σ = 5000
medium σ = 50
high σ = 5

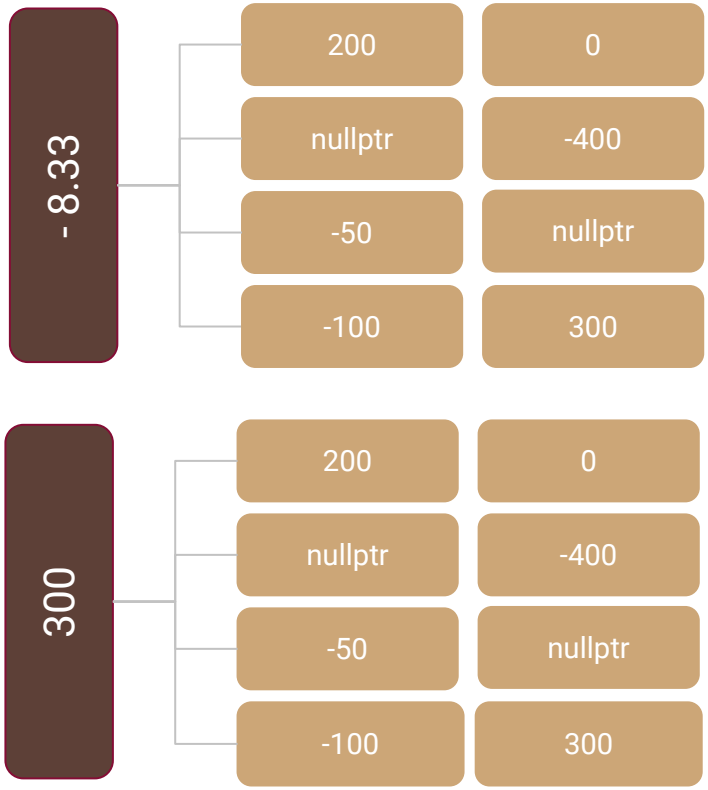
Sonar data will more closely resemble the dense model

Fig 10. Inserting values into a tree with check and low locality.

Updating Log Odds - Strategy

Mean of children:
- Minimizes error
- Optimistic approach
- May lower probability of node being empty while it is actually full

Max of children:
- Pessimistic approach
- Lowers odds of the UAV blindly going into occupied areas



Raycast Analysis

Bresenham vs DDA - Brief Overview

- Bresenham's line algorithm:
  - Travels in all axis at once
  - Handles diagonal transitions very well
  - Very accurate
- Digital Differential Analyzer (DDA)
  - Travels in only one axis at a time
  - Goes in the direction of the closest axis
  - May miss the target (see next slide)

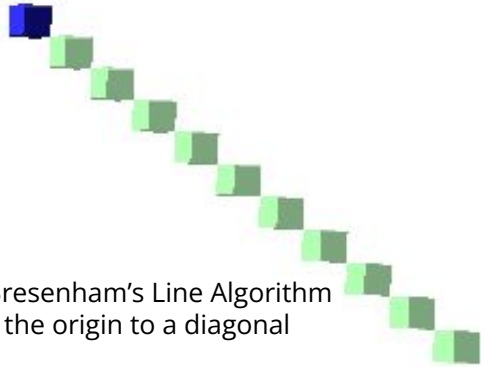


Fig 11. Bresenham's Line Algorithm from the origin to a diagonal

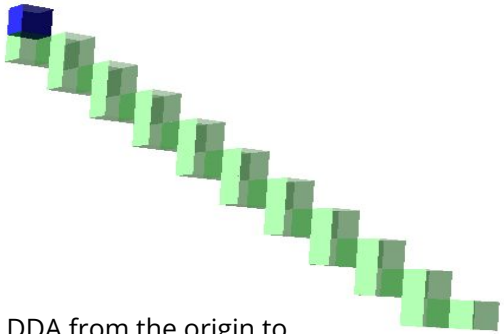


Fig 12. DDA from the origin to a diagonal

DDA misses

- DDA may miss the target
  - Final portion of the beam may end up missing or off target.
- Traditionally has 2 different stopping conditions
  a. Current node is the destination (rare)
  b. The distance travelled is larger than the distance between the start and end points, so we already travelled the required distance and should stop, as we are off-track and would never reach the end
- Open source octomap implementation also suffers from these misses

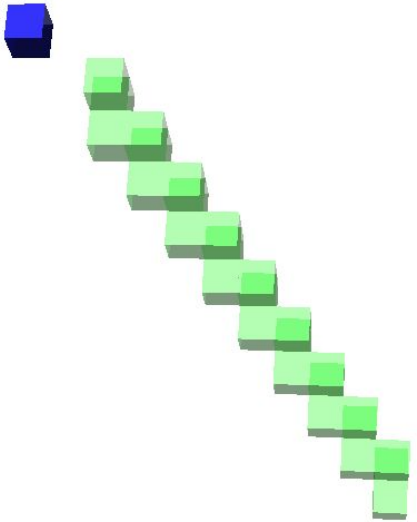


Fig 13. DDA raycast missing from (0, 0, 0) to (1, 1, 1)

DDA misses - Finishing the raycast

- Third custom condition, applied after the others fail:
  - When we stop early, we are guaranteed to be within the neighborhood of the target, thanks to DDA
  - We snap a final path towards the destination
- Sometimes causes distortions in the end of the rays
  - But ensures we mark as empty all required the nodes until the destination
- Is computationally expensive

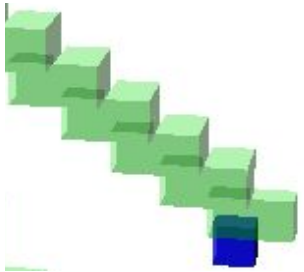


Fig 14. Example of distortion at the end of the rays.

Bresenham vs DDA - Tradeoffs

- Bresenham’s accuracy may not always be desirable;
  - Since it is similar to a laser, it does not represent the broader range of voxels the sonar scans.
- Bresenham is more efficient than DDA and has simpler operations;
  - Additions/subtractions rather than multiplications/divisions.

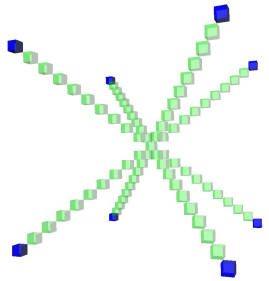


Fig 15. Bresenham’s Line Algorithm from the origin to all diagonals

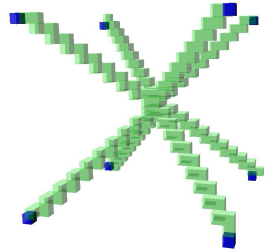


Fig 16. DDA from the origin to all diagonals

Comparing results

- Parallelism obtained using OpenMP library.
- A ray was cast from the origin, (0,0,0), to every node of the plane, a total of 20 991 638 nodes

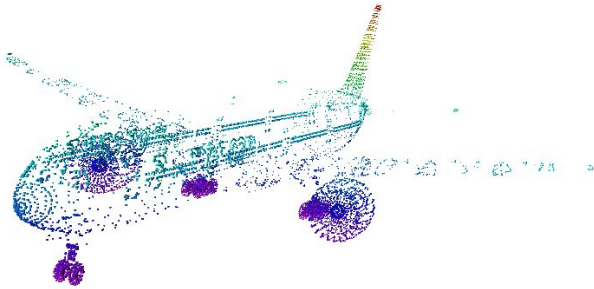
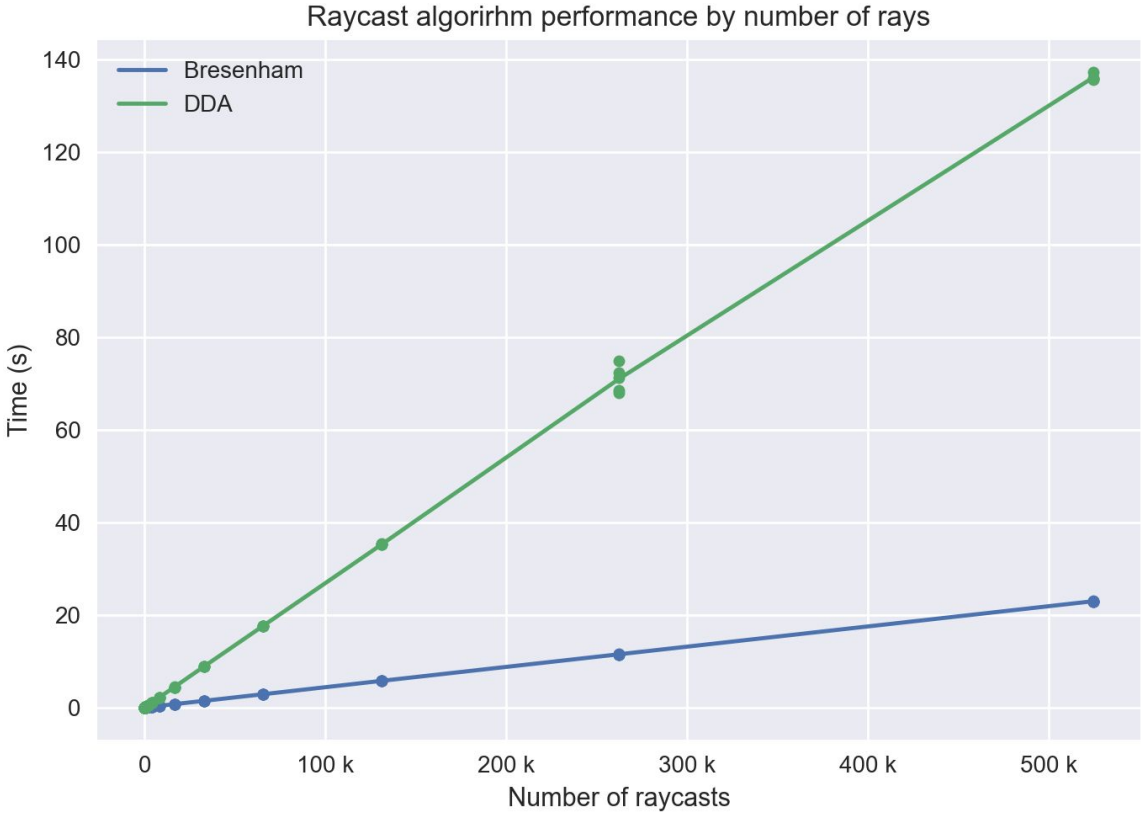


Fig 17. Plane model obtained using the octomap implementation and ModelNet40's point cloud

	Bresenham's Line Algorithm	DDA
Average Time With Parallelism	23.2s	35.3s
Average Time With No Parallelism	26.2s	63.3s

Comparing Results



For traditional applications, DDA would be preferable, as we want to find the first collision

For our use case, we only want the path from the origin to the target, so Bresenham's performance and lack of misses indicate it will be the most beneficial to our project

Fig 18. Raycast algorithm performance comparison

Gaussian Filter Analysis

Gaussian Filter

- Applies blur to the image in order to try to reduce the image noise.
- Improves edge detection.
- Applying the blur to a sweep, which contains 399 beams, takes 1 millisecond.

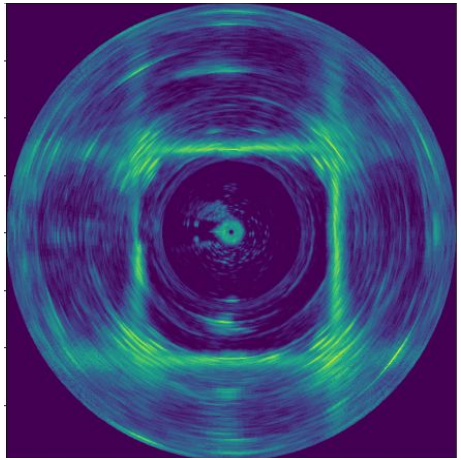


Fig 19. No filter

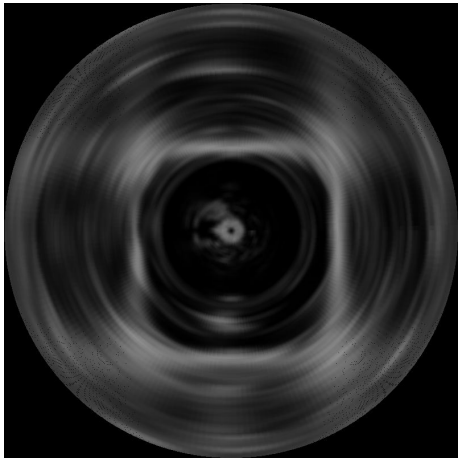
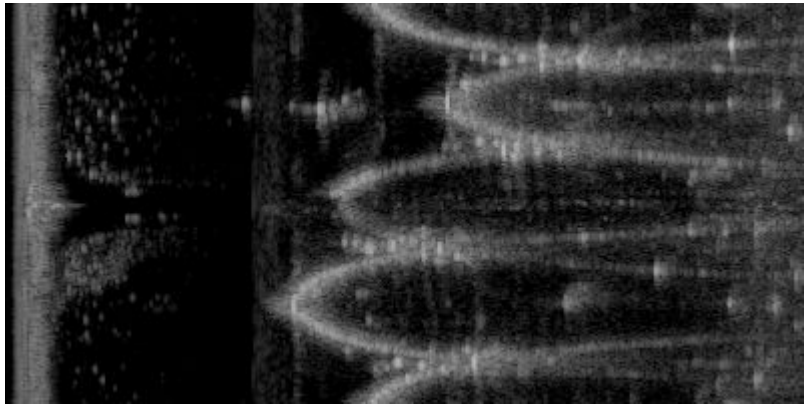


Fig 20. Gaussian filter

Gaussian Filter Parameter Tuning



- Converting beam data into cartesian coordinates is too demanding
- Applying the algorithms over the polar images proved to be reliable

Fig 21. No filter polar coordinates

After some experimentation, the following parameters yielded good results:

- Sigma 2.3
- Kernel size 11

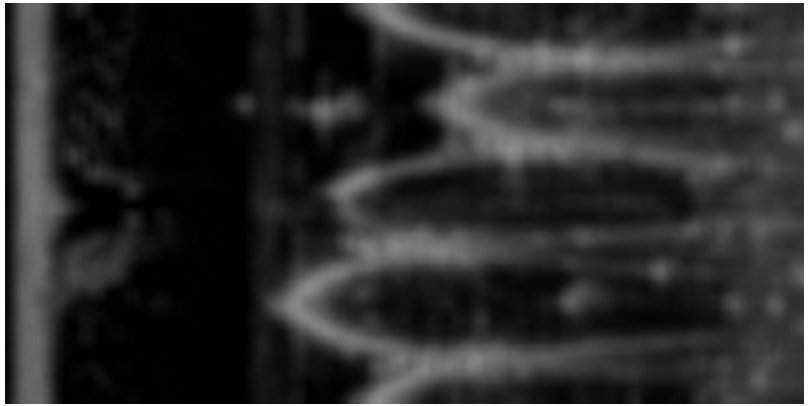


Fig 22. Gaussian filter polar coordinates

Edge Detection: First Approach

Applied over a Gaussian filter with kernel 11 and sigma 2.3

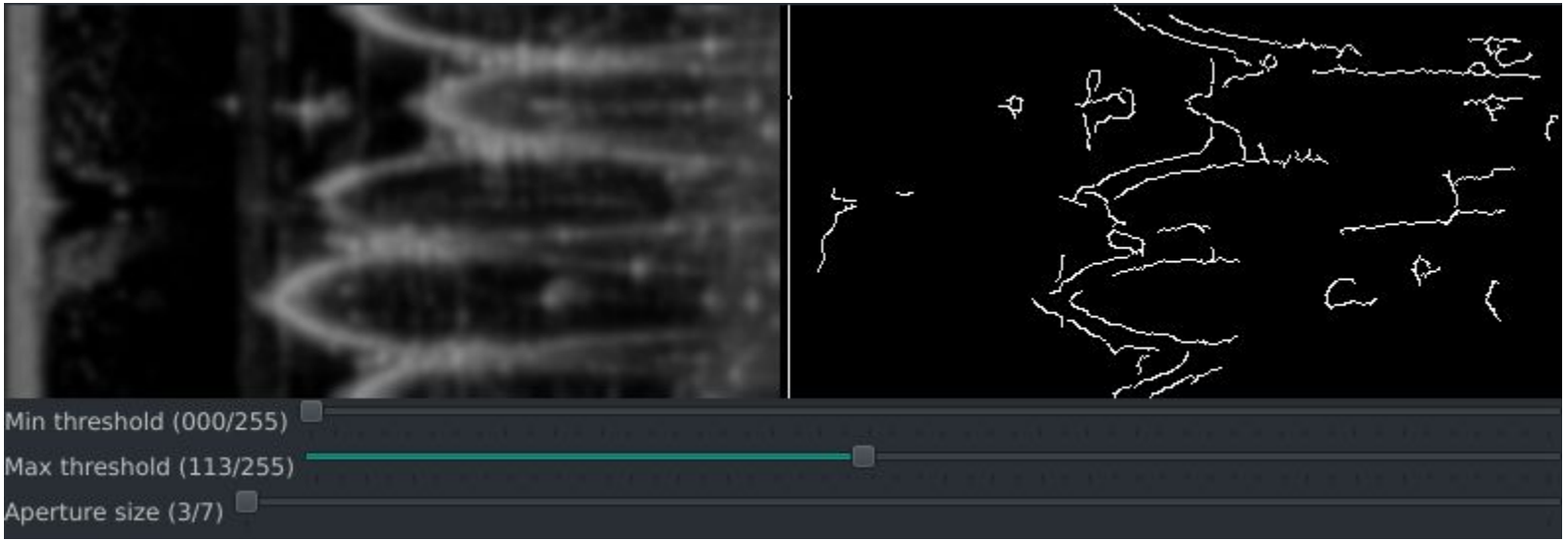


Fig 23. Canny algorithm for edge detection

Questions?

Slides